Teste12_P_IE509A_2021S2_AndresFelipeEscallonPortilla_Doutorado_FEEC_Unicamp

Testinho (Dez-2021)

Na cadeia de Markov abaixo, qual a matriz de transição P? Quais as probabilidades estacionárias? Outra pergunta: eu sei que eu saí do estado 1 no instante n. Qual a probabilidade de eu ter ido para estado 2?

Cadeia de Markov: veja as graficas abaixo ou neste link 1 retirado deste link no overleaf compartilhado por a minha colega Fernanda Caldas ("...*quem quiser a cadeia de markov do enunciado em latex*...") da disciplina do **Processos Estocáticos** ou neste link 2, e como fazela neste meu repo baseado em isto (por NaysanSaran)

```
# Para poder cargar archivos desde Google Drive
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

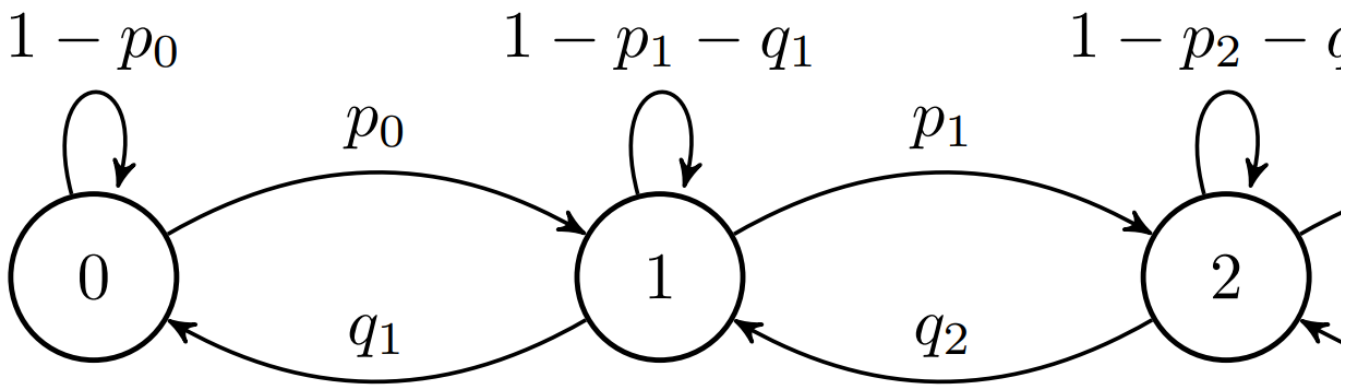
```
from google.colab import files
uploaded = files.upload()
```

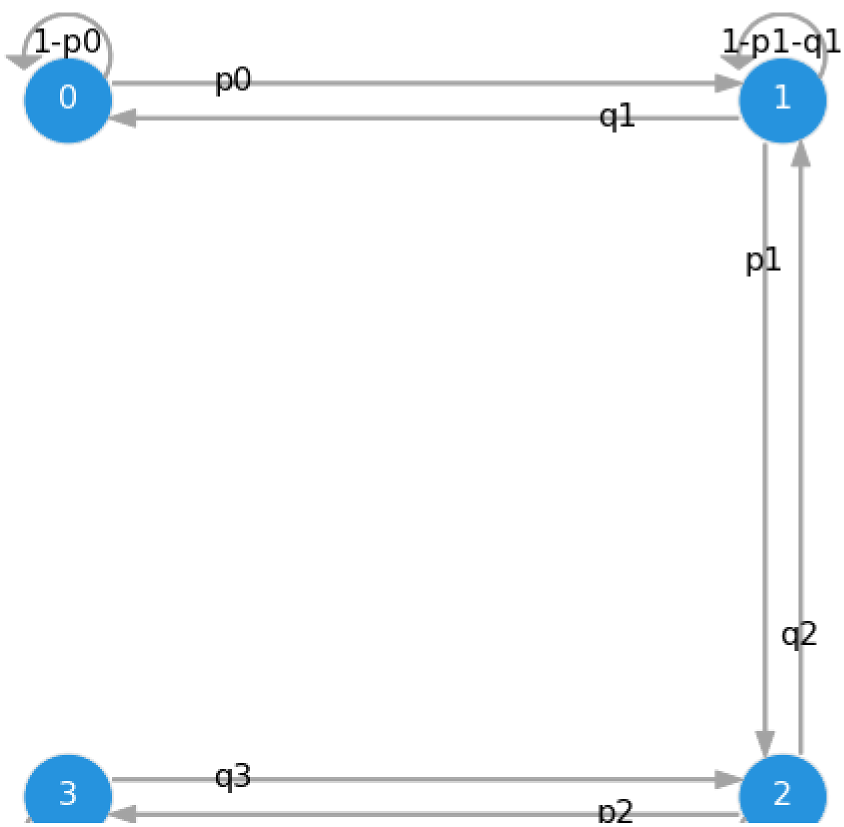
Elegir archivos 2 archivos

- **birth_death_markov_chain.png**(image/png) - 74825 bytes, last modified: 7/12/2021 - 100% done
- **markov-chain-four-states-my_example.png**(image/png) - 21904 bytes, last modified: 7/12/2021 - 100%

```
Saving birth_death_markov_chain.png to birth_death_markov_chain (1).png
Saving markov-chain-four-states-my_example.png to markov-chain-four-states-my_example (:
```

```
from IPython.display import Image, display
display(Image('birth_death_markov_chain.png'))
display(Image('markov-chain-four-states-my_example.png'))
```






```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec  1 22:27:53 2021

@author: Administrador (Andrés Felipe Escallón Portilla)
"""
#############REFERENCES################################
'''
https://claudiovz.github.io/scipy-lecture-notes-ES/packages/sympy.html
https://docs.sympy.org/latest/modules/vector/index.html
https://jorgedelossantos.github.io/apuntes-python/SymPy.html
https://www.matesfacil.com/matrices/metodo-matriz-inversa-resolver-sistemas-ecuaciones-lineal
https://ernestocrespo13.wordpress.com/2015/02/21/resolucion-de-sistemas-de-ecuaciones-con-sym
http://research.iac.es/sieinvens/python-course/sympy.html
https://relopezbriega.github.io/blog/2015/06/14/algebra-lineal-con-python/
```

```
https://www.superprof.es/diccionario/matematicas/algebralineal/regla-cramer.html
https://es.acervolima.com/2021/02/09/python-sympy-metodo-matrix-eigenvects/
https://pythondiario.com/2019/01/matrices-en-python-y-numpy.html
https://yosoytuprofe.20minutos.es/2016/10/18/potencia-n-esima-de-una-matriz/
https://www.i-ciencias.com/pregunta/73686/una-matriz-a-la-potencia-de-cero-da-matriz-identida
http://pyscience-brasil.wikidot.com/docitem:numpy-identity
https://www.delftstack.com/pt/api/numpy/python-numpy-linalg.inverse/
https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html
https://github.com/platzi/algebra-lineal-python
https://github.com/platzi/algebra-lineal-python/blob/master/09%20-%20Transformaciones%20Linea
https://interactivechaos.com/es/python/scenario/inversion-de-un-array-numpy
https://www.vooo.pro/insights/wp-content/uploads/2018/04/Vooo-Insights-Python_numpy_acesso_ra
https://runebook.dev/es/docs/numpy/reference/generated/numpy.diag
https://www.stat.auckland.ac.nz/~fewster/325/notes/ch8.pdf
https://www.youtube.com/watch?v=8cXG5lF7pvA
https://www.youtube.com/watch?v=13DHglQag0g&t=126s
https://www.youtube.com/watch?v=4W9a0J69Oqg
https://ece307.cankaya.edu.tr/uploads/files/introduction%20to%20probability%20(bertsekas,%202
http://blog.espol.edu.ec/estg1003/tag/cadenas-markov/
'''
#######################################################

import sympy as sp
from sympy import *
from sympy import Matrix
from sympy import Symbol
#from sympy.matrices import Matrix

#******************************************************
print('Testing first...')
print(Matrix([[1,0], [0,1]]))

x = Symbol('x')
y = Symbol('y')

A = Matrix([[1,x], [y,1]])

print(A)
print(A**2)
#******************************************************

print()
print("birth_death_markov_chain:")
print()

#working symbollically with sympy

p0 = Symbol('p0')
p1 = Symbol('p1')
p2 = Symbol('p2')
p3 = Symbol('p3')
```

```
q1 = Symbol('q1')
q2 = Symbol('q2')
q3 = Symbol('q3')
#(p0,p1,p2,q1,q2,q3) = symbols("p0,p1,p2,q1,q2,q3") # it also works this way

#pi0 = Symbol('pi0')
#pi1 = Symbol('pi1')
#pi2 = Symbol('pi2')
#pi3 = Symbol('pi3')
(pi0,pi1,pi2,pi3) = symbols("pi0,pi1,pi2,pi3")


#transition matrix
P = Matrix( [ [1-p0, p0, 0, 0],  [q1, 1-p1-q1, p1, 0], [0, q2, 1-p2-q2, p2], [0, 0, q3, 1-q3]
print('\n P:\n', P)
print('\n det(P):\n', P.det())

#pi (row vector)
Pi = Matrix( [pi0,pi1,pi2,pi3] ).T #row vector = column vector transposed
print('\nPi:\n',Pi)

PiP = Pi * P
print('\nPiP:\n',PiP)

eigenvalue = 1


#res = Pi * (P.inv()) # this is not the solution because it does not involve the relationship
#res0 = Pi * (P.inv()) # this is not the solution because it does not involve the relationshi


#Mmod = Matrix( [ [1-p0-eigenvalue, p0, 0, 0],  [q1, 1-p1-q1-eigenvalue, p1, 0], [0, q2, 1-p2
Mmod = Matrix([[-p0, q1, 0, 0], [p0, -p1 - q1, q2, 0], [0, p1, -p2 - q2, q3], [1, 1, 1, 1]])
print('\n Mmod= \n',Mmod)
print('\n det(Mmod)= \n',Mmod.det()) #**se puede con la inversa (det=!0) y también con solve(

#PiMmod = Pi * Mmod
PiMmod =  Mmod * Pi.T
#PiMmod = Matrix([[-p0*pi0 + q1*pi1, p0*pi0 -(p1 + q1)*pi1 + q2*pi2, p1*pi1 -(p2 + q2)*pi2 +

print('\nPiMmod:\n',PiMmod)

#b = Matrix([0,0,0,1]).T #row vector = column vector transposed
#b = Matrix([0,0,0,pi3]).T #row vector = column vector transposed
b = Matrix([0,0,0,1]).T #row vector = column vector transposed
print('\n b= \n',b)

Mmod_inv = simplify(Mmod.inv())
print('\n Mmod_inv= \n',Mmod_inv)
#pimod = Mmod.inv() * b.T #**
```

```
#pimod = Mmod.inv() * b.T #**
#pimod = b.T * Mmod_inv  #**
#pimod = simplify(b * Mmod_inv)  #**
pimod = simplify(Mmod_inv * b.T)  #**
print('\n pimod= \n',simplify(pimod))

#xmod = np.linalg.solve(Mmod, b.transpose())
#print('\n xmod= \n',xmod)

PiMmod2 = Matrix([-p0*pi0 + pi1*q1, p0*pi0 + pi1*(-p1 - q1) + pi2*q2, p1*pi1 + pi2*(-p2 - q2)
PiMmod3 = Matrix([[-p0*pi0 + pi1*q1, p0*pi0 + pi1*(-p1 - q1) + pi2*q2, p1*pi1 + pi2*(-p2 - q2
#PiMmod4 = Matrix([[-p0*pi0 + pi1*q1 - 0, p0*pi0 + pi1*(-p1 - q1) + pi2*q2 - 0, p1*pi1 + pi2*
PiMmod4 = Matrix([-p0*pi0 + pi1*q1 - 0, p0*pi0 + pi1*(-p1 - q1) + pi2*q2 - 0, p1*pi1 + pi2*(-

#result0 = solve(PiMmod) #FUNCIONO!!!! (con inversa no se porque no funcionó aquí, tal vez po
#result0 = solve(PiMmod.T,Matrix([pi0,pi1,pi2,pi3]).T) #FUNCIONO!!!! (con inversa no se porqu
#result0 = solve(PiMmod,[pi0,pi1,pi2,pi3]) #FUNCIONO!!!! (con inversa no se porque no funcion
#result0 = solve(PiMmod,Matrix([pi0,pi1,pi2,pi3]).T) #FUNCIONO!!!! (con inversa no se porque
#result0 = solve(PiMmod2.T,Matrix([pi0,pi1,pi2,pi3]).T) #FUNCIONO!!!! (con inversa no se porq
#result = solve( [ pi0 - pi3*q1*q2*q3/(p0*p1*p2), pi1 - pi3*q2*q3/(p1*p2), pi2 - pi3*q3/p2, p
#result0 = solve(PiMmod3,[pi0,pi1,pi2,pi3]) #FUNCIONO!!!! (con inversa no se porque no funcio
result0 = solve(PiMmod4,[pi0,pi1,pi2,pi3]) #FUNCIONO!!!! (con inversa TAMBIÉN FUNCIONAAAA!)
print('\n result0 = \n', result0) #this solution involves the complementary equation where su
#print('\n result0 = \n', simplify(result0)) #this solution involves the complementary equati

#pi3 = 1 - (pi0 + pi1 + pi2)
result = solve( [ -p0*pi0 + q1*pi1, p0*pi0 - (p1+q1)*pi1 + q2*pi2, p1*pi1 - (p2+q2)*pi2 + q3*
print('\n result = \n', result) # this is the trivial solution (x=0)


##WORKING:
print()
print ("WORKING from here:")

print ("Real way to get the eigenvalue for lambda = 1 :")

print("\n P transposed: \n", P.T)
Pt_minus_1I = P.T - sp.eye(4)
print('\n Pt_minus_1I:\n', Pt_minus_1I) # real way to get the eigenvalue for lambda = 1

print("\n Pi transposed: \n", Pi.T)
Pt_minus_1I_by_Pit = Pt_minus_1I * Pi.T
print('\n Pt_minus_1I_by_Pit : \n', Pt_minus_1I_by_Pit)

res_0 = solve(Pt_minus_1I_by_Pit, Pi.T) #it returns the eigenvector for the eigenvale of lamb
print('\n res_0 = \n', res_0) # eigenvalues: this is the solution of (A-lambda*I)*x=0, with l

##############################################################################################
print("\n Another way to do it more automatically:\n")

tam = int(len(P)**0.5) #number of rows/columns
```

```
#arranging and solving the linear system to get the stationary propabilities
Pt_minus_1I = P.T - eigenvalue*sp.eye(tam)
print('\n Pt_minus_1I:\n', Pt_minus_1I)

Pt_minus_1I[-1,:] = sp.ones(tam)[-1,:] #replacing the last row with ones (probability constra

Pt_minus_1I_replaced = Pt_minus_1I

b = sp.zeros(tam)[-1,:] #independent row vector (zeros)
b[-1] = 1  #replacing the last row with one (probability constraint)
b_replaced = b

Pt_minus_1I_replaced_by_Pit = Pt_minus_1I_replaced * Pi.T
print('\n Pt_minus_1I_replaced_by_Pit : \n', Pt_minus_1I_replaced_by_Pit)

Pn_calc = solve(Pt_minus_1I_replaced_by_Pit - b_replaced.T , Pi.T) #solving the resulting lin

print('\n Stationary probabilities: \n')
print(' Pn_calc =', pretty(Pn_calc)) #pretty used to show it more beautifully
###########################################################################################

print ("Another way to get the eigenvalue for lambda = 1 (without using the transposed):")

P_minus_1I = P - sp.eye(4)
print('\n det(P_minus_1I):\n', P_minus_1I.det()) # det = 0 to get a non-trivial solution

Pi_by_P_minus_1I = Pi * P_minus_1I

print('\n Pi_by_P_minus_1I : \n', Pi_by_P_minus_1I)

#res = solve([3*x+9*y-10*z-24,x-6*y+4*z+4,10*x-2*y+8*z-20],[x,y,z])
res = solve(Pi_by_P_minus_1I,[pi0,pi1,pi2,pi3]) #it returns the eigenvector for the eigenvale
print('\n res = \n', res) # eigenvalues: this is the solution of (A-lambda*I)*x=0, with lambd

suma_x = q1*q2*q3/(p0*p1*p2) + q2*q3/(p1*p2) + q3/p2 + 1
print('\n suma_x = ', suma_x)

x_normalizado = (1/suma_x) * Matrix([ q1*q2*q3/(p0*p1*p2), q2*q3/(p1*p2), q3/p2, 1])#FUNCIONÓ
print('\n x_normalizado \n =', simplify(x_normalizado)) #FUNCIONÓ!!!

#suma_x2 = res[pi0][4:] + res[pi1][4:] + res[pi2][4:] + 1
suma_x2 = res[pi0]/pi3 + res[pi1]/pi3 + res[pi2]/pi3 + pi3/pi3
x_normalizado2 = (1/suma_x2) * Matrix([ res[pi0]/pi3 , res[pi1]/pi3 , res[pi2]/pi3, pi3/pi3])
print('\n x_normalizado2 \n =', simplify(x_normalizado2)) #FUNCIONÓ!!!


res2 = solve( [ pi0 - pi3*q1*q2*q3/(p0*p1*p2), pi1 - pi3*q2*q3/(p1*p2), pi2 - pi3*q3/p2, pi0
print('\n res2 = \n', res2) #this solution involves the complementary equation where sum_j of

pi3 = 1 - (pi0 + pi1 + pi2)
```

```
res3 = solve( [ -p0*pi0 + q1*pi1, p0*pi0 - (p1+q1)*pi1, p1*pi1 - (p2-q2)*pi2 + q3*pi3, p2*pi2
print('\n res3 = \n', res3) # this is the trivial solution (x=0): useless


print('\n Testing manually \n:')

sum = (q1*q2*q3/(p0*p1*p2)) + (q2*q3/(p1*p2)) + (q3/p2) + (1)

stationary_vector = (1/sum) * Matrix([q1*q2*q3/(p0*p1*p2), q2*q3/(p1*p2), q3/p2,1])

print('\n stationary_vector \n:', stationary_vector)

print('\n stationary_vector simplified \n:', simplify(stationary_vector))

print("\n stationary_vector components must add to 1 (Probabilities): \n", simplify(stationar


#################################################################################
print()
print("Now trying with eigenvectors automatically:")

M = Matrix([[4,2],[3,3]])
print("Matrix : {} ".format(M))
M_eigenvects = M.eigenvects()
print("Eigenvects of a matrix : {}".format(M_eigenvects))

print()

#print("Matrix : {} ".format(P))
#P_eigenvects = P.eigenvects()  #muy pesado computacionalmente y no se ve la convergencia fác
#print("Eigenvects of a matrix : {}".format(P_eigenvects))

#print('\n P^10 \n=', simplify(P*P*P*P*P*P*P*P*P*P))#muy pesado computacionalmente y no se ve
```

Pn_calc =

$$\left\{ \pi_0: \frac{q_1 \cdot q_2 \cdot q_3}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3}, \pi_1: \frac{p_0 \cdot q_2 \cdot}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3}, \pi_2: \frac{p_0 \cdot p_1 \cdot q_3}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3}, \pi_3: \frac{p_0 \cdot p_1 \cdot p_2}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3} \right\}$$

```
Another way to get the eigenvalue for lambda = 1 (without using the transposed):

 det(P_minus_1I):
 0

 Pi_by_P_minus_1I :
 Matrix([[-p0*pi0 + pi1*q1, p0*pi0 + pi1*(-p1 - q1) + pi2*q2, p1*pi1 + pi2*(-p2 - q2
```

```
 res =
 {pi0: pi3*q1*q2*q3/(p0*p1*p2), pi1: pi3*q2*q3/(p1*p2), pi2: pi3*q3/p2}

 suma_x =  1 + q3/p2 + q2*q3/(p1*p2) + q1*q2*q3/(p0*p1*p2)

 x_normalizado
 = Matrix([[q1*q2*q3/(p0*p1*p2 + p0*p1*q3 + p0*q2*q3 + q1*q2*q3)], [p0*q2*q3/(p0*p1*

 x_normalizado2
 = Matrix([[q1*q2*q3/(p0*p1*p2 + p0*p1*q3 + p0*q2*q3 + q1*q2*q3)], [p0*q2*q3/(p0*p1*

 res2 =
 {pi0: q1*q2*q3/(p0*p1*p2 + p0*p1*q3 + p0*q2*q3 + q1*q2*q3), pi1: p0*q2*q3/(p0*p1*p2

 res3 =
 {pi0: 0, pi1: 0, pi2: 0, -pi0 - pi1 - pi2 + 1: 0}

 Testing manually
:

 stationary_vector
: Matrix([[q1*q2*q3/(p0*p1*p2*(1 + q3/p2 + q2*q3/(p1*p2) + q1*q2*q3/(p0*p1*p2)))], [

 stationary_vector simplified
: Matrix([[q1*q2*q3/(p0*p1*p2 + p0*p1*q3 + p0*q2*q3 + q1*q2*q3)], [p0*q2*q3/(p0*p1*p

 stationary_vector components must add to 1 (Probabilities):
 1

Now trying with eigenvectors automatically:
Matrix : Matrix([[4, 2], [3, 3]])
Eigenvects of a matrix : [(1, 1, [Matrix([
[-2/3],
[   1]])]), (6, 1, [Matrix([
[1],
[1]])])]
```

RESPOSTAS:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Dec  8 07:57:00 2021

@author: Administrador (Andrés Felipe Escallón Portilla)
"""
############REFERENCE################################

#http://blog.espol.edu.ec/estg1003/tag/cadenas-markov/
```

```
#######################################################
import sympy as sp
from sympy import *
from sympy import Matrix
from sympy import Symbol
############################################################################################
print("birth_death_markov_chain:")

#working symbollically with sympy
(p0,p1,p2,q1,q2,q3) = symbols("p0,p1,p2,q1,q2,q3")
(pi0,pi1,pi2,pi3) = symbols("pi0,pi1,pi2,pi3")

#transition matrix
P = Matrix( [ [1-p0, p0, 0, 0],  [q1, 1-p1-q1, p1, 0], [0, q2, 1-p2-q2, p2], [0, 0, q3, 1-q3]
print('\n Qual a matriz de transição P? \n')
print('\n Transition matrix: \n')
print('\n P:\n', P)

#pi (row vector)
Pi = Matrix( [pi0,pi1,pi2,pi3] ).T #row vector = column vector transposed
print('\nPi:\n',Pi)

eigenvalue = 1

############################################################################################
print("\n Another way to do it more automatically:\n")

tam = int(len(P)**0.5) #number of rows/columns

#arranging and solving the linear system to get the stationary propabilities
Pt_minus_1I = P.T - eigenvalue*sp.eye(tam)
print('\n Pt_minus_1I:\n', Pt_minus_1I)

Pt_minus_1I[-1,:] = sp.ones(tam)[-1,:] #replacing the last row with ones (probability constra

Pt_minus_1I_replaced = Pt_minus_1I

b = sp.zeros(tam)[-1,:] #independent row vector (zeros)
b[-1] = 1  #replacing the last row with one (probability constraint)
b_replaced = b

Pt_minus_1I_replaced_by_Pit = Pt_minus_1I_replaced * Pi.T
print('\n Pt_minus_1I_replaced_by_Pit : \n', Pt_minus_1I_replaced_by_Pit)

Pn_calc = solve(Pt_minus_1I_replaced_by_Pit - b_replaced.T , Pi.T) #solving the resulting lin

print('\n Quais as probabilidades estacionárias?:\n')
print('\n Stationary probabilities: \n')
print(' Pn_calc =', pretty(Pn_calc)) #pretty used to show it more beautifully
```

```
print('\n Eu sei que eu saí do estado 1 no instante n, qual a probabilidade de eu ter ido par
resposta="P(X_{n+1} = 2 | X_{n+1} ≠ 1 , X_{n} = 1 )} = P(X_{n+1} = 2 , X_{n+1}  ≠ 1 | X_{n} =
print(pretty(resposta))
###########################################################################################
```

```
birth_death_markov_chain:

 Qual a matriz de transição P?


 Transition matrix:


 P:
 Matrix([[1 - p0, p0, 0, 0], [q1, -p1 - q1 + 1, p1, 0], [0, q2, -p2 - q2 + 1, p2], [0, 0

Pi:
 Matrix([[pi0, pi1, pi2, pi3]])

 Another way to do it more automatically:


 Pt_minus_1I:
 Matrix([[-p0, q1, 0, 0], [p0, -p1 - q1, q2, 0], [0, p1, -p2 - q2, q3], [0, 0, p2, -q3]]

 Pt_minus_1I_replaced_by_Pit :
 Matrix([[-p0*pi0 + pi1*q1], [p0*pi0 + pi1*(-p1 - q1) + pi2*q2], [p1*pi1 + pi2*(-p2 - q2

 Quais as probabilidades estacionárias?:


 Stationary probabilities:
```

Pn_calc =
$$\left\{ \pi_0: \frac{q_1 \cdot q_2 \cdot q_3}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3},\ \pi_1: \frac{p_0 \cdot q_2 \cdot q_3}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3},\ \pi_2: \frac{p_0 \cdot p_1 \cdot q_3}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3},\ \pi_3: \frac{p_0 \cdot p_1 \cdot p_2}{p_0 \cdot p_1 \cdot p_2 + p_0 \cdot p_1 \cdot q_3 + p_0 \cdot q_2 \cdot q_3 + q_1 \cdot q_2 \cdot q_3} \right\}$$

```
 Eu sei que eu saí do estado 1 no instante n, qual a probabilidade de eu ter ido para es

P(X_{n+1} = 2 | X_{n+1} ≠ 1 , X_{n} = 1 )} = P(X_{n+1} = 2 , X_{n+1}  ≠ 1 | X_
{n} = 1 ) / P(X_{n+1} ≠ 1 , X_{n} = 1 ) = P(X_{n+1} = 2 | X_{n} = 1 ) / P(X_{n
+1} ≠ 1 , X_{n} = 1 ) = p_{12} / 1-p_{11} = p_1 / 1-[1-(q_{1}+p_{1})] = p_{1}
/ 1-1+(q_{1}+p_{1}) = p_{1} / q_{1}+p_{1}
```

Resposta (Qual a probabilidade de eu ter ido para estado 2?):

Equações em Markdown: https://programmerclick.com/article/9139292621/

$$P(X_{n+1} = 2 | X_{n+1} \neq 1, X_n = 1) =$$
$$= \frac{P(X_{n+1}=2, X_{n+1} \neq 1 | X_n = 1)}{P(X_{n+1} \neq 1, X_n = 1)} =$$
$$= \frac{P(X_{n+1}=2 | X_n = 1)}{P(X_{n+1} \neq 1, X_n = 1)}$$
$$= \frac{p_{12}}{1-p_{11}}$$
$$= \frac{p_1}{1-[1-(q_1+p_1)]}$$
$$= \frac{p_1}{1-1+(q_1+p_1)}$$
$$= \frac{p_1}{q_1+p_1}$$

Mais referências:

https://github.com/maximtrp/mchmm

http://www.blackarbs.com/blog/introduction-hidden-markov-models-python-networkx-sklearn/2/9/2017

https://github.com/drvinceknight/unpeudemath/blob/gh-pages/assets/code/Visualising%20Markov%20Chains.ipynb

0 s se ejecutó 09:55